School of Data and Computer Science, Sun Yat-sen University

E13 BP Algorithm (C++/Python)

16110917 Zhaoshuai Liu

November 28, 2018

Contents

1 Horse Colic Data Set

The description of the horse colic data set (`http://archive.ics.uci.edu/ml/datasets/Horse+Colic`) is as follows:

Data Set Characteristics:	Multivariate	**Number of Instances:**	368	**Area:**	Life
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	27	**Date Donated**	1989-08-06
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	108569

We aim at trying to predict if a horse with colic will live or die.

Note that we should deal with missing values in the data! Here are some options:

- Use the features mean value from all the available data.
- Fill in the unknown with a special value like -1.
- Ignore the instance.
- Use a mean value from similar items.
- Use another machine learning algorithm to predict the value.

2 Reference Materials

1. Stanford: **CS231n: Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li,etc.
   - Course website: `http://cs231n.stanford.edu/2017/syllabus.html`
   - Video website: `https://www.bilibili.com/video/av17204303/?p=9&tdsourcetag=s_pctim_aiomsg`
2. **Machine Learning** by Hung-yi Lee
   - Course website: `http://speech.ee.ntu.edu.tw/~tlkagk/index.html`
   - Video website: `https://www.bilibili.com/video/av9770302/from=search`
3. A Simple neural network code template

```
# -*- coding: utf-8 -*
import random
import math

# Shorthand:
# "pd_" as a variable prefix means "partial derivative"
# "d_" as a variable prefix means "derivative"
# "_wrt_" is shorthand for "with respect to"
```

```
# "w_ho" and "w_ih" are the index of weights from hidden to output layer neurons and input to hidden layer neurons respectively

class NeuralNetwork:
    LEARNING_RATE = 0.5
    def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights = None, hidden_layer_bias = None, output_layer_weights = None, output_layer_bias = None):
    #Your Code Here

    def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
    #Your Code Here

    def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self, output_layer_weights):
    #Your Code Here

    def inspect(self):
        print('------')
        print('* Inputs: {}'.format(self.num_inputs))
        print('------')
        print('Hidden Layer')
        self.hidden_layer.inspect()
        print('------')
        print('* Output Layer')
        self.output_layer.inspect()
        print('------')

    def feed_forward(self, inputs):
        #Your Code Here

    # Uses online learning, ie updating the weights after each training case
    def train(self, training_inputs, training_outputs):
        self.feed_forward(training_inputs)

        # 1. Output neuron deltas
        #Your Code Here
        #   E/   z

        # 2. Hidden neuron deltas
```

```
45          # We need to calculate the derivative of the error with respect to the
                output of each hidden layer neuron
46          # dE/dy = E/z * z/y = E/z * w
47          #   E/z = dE/dy * z/
48          #Your Code Here
49
50          # 3. Update output neuron weights
51          #   E/w = E/z * z/w
52          #  w = * E/w
53          #Your Code Here
54
55          # 4. Update hidden neuron weights
56          #   E/w = E/z * z/w
57          #  w = * E/w
58          #Your Code Here
59
60      def calculate_total_error(self, training_sets):
61          #Your Code Here
62          return total_error
63
64  class NeuronLayer:
65      def __init__(self, num_neurons, bias):
66
67          # Every neuron in a layer shares the same bias
68          self.bias = bias if bias else random.random()
69
70          self.neurons = []
71          for i in range(num_neurons):
72              self.neurons.append(Neuron(self.bias))
73
74      def inspect(self):
75          print('Neurons:', len(self.neurons))
76          for n in range(len(self.neurons)):
77              print(' Neuron', n)
78              for w in range(len(self.neurons[n].weights)):
79                  print('  Weight:', self.neurons[n].weights[w])
80              print('  Bias:', self.bias)
81
82      def feed_forward(self, inputs):
83          outputs = []
84          for neuron in self.neurons:
```

```
85              outputs.append(neuron.calculate_output(inputs))
86          return outputs
87
88      def get_outputs(self):
89          outputs = []
90          for neuron in self.neurons:
91              outputs.append(neuron.output)
92          return outputs
93
94  class Neuron:
95      def __init__(self, bias):
96          self.bias = bias
97          self.weights = []
98
99      def calculate_output(self, inputs):
100     #Your Code Here
101
102     def calculate_total_net_input(self):
103     #Your Code Here
104
105     # Apply the logistic function to squash the output of the neuron
106     # The result is sometimes referred to as 'net' [2] or 'net' [1]
107     def squash(self, total_net_input):
108     #Your Code Here
109
110     # Determine how much the neuron's total input has to change to move closer to
            the expected output
111     #
112     # Now that we have the partial derivative of the error with respect to the
            output ( E/ y ) and
113     # the derivative of the output with respect to the total net input (dy /
            dz ) we can calculate
114     # the partial derivative of the error with respect to the total net input.
115     # This value is also known as the delta ( ) [1]
116     # = E/ z = E/ y * dy /dz
117     #
118     def calculate_pd_error_wrt_total_net_input(self, target_output):
119     #Your Code Here
120
121     # The error for each neuron is calculated by the Mean Square Error method:
122     def calculate_error(self, target_output):
```

```
123     #Your Code Here
124
125     # The partial derivate of the error with respect to actual output then is
            calculated by:
126     # = 2 * 0.5 * (target output - actual output) ^ (2 - 1) * -1
127     # = -(target output - actual output)
128     #
129     # The Wikipedia article on backpropagation [1] simplifies to the following,
            but most other learning material does not [2]
130     # = actual output - target output
131     #
132     # Alternative, you can use (target - output), but then need to add it during
            backpropagation [3]
133     #
134     # Note that the actual output of the output neuron is often written as y
            and target output as t so:
135     # = E/ y = -(t - y)
136     def calculate_pd_error_wrt_output(self, target_output):
137     #Your Code Here
138
139     # The total net input into the neuron is squashed using logistic function to
            calculate the neuron's output:
140     # y = = 1 / (1 + e^(-z))
141     # Note that where represents the output of the neurons in whatever layer
            we're looking at and represents the layer below it
142     #
143     # The derivative (not partial derivative since there is only one variable) of
             the output then is:
144     # dy/dz = y * (1 - y)
145     def calculate_pd_total_net_input_wrt_input(self):
146     #Your Code Here
147
148     # The total net input is the weighted sum of all the inputs to the neuron and
             their respective weights:
149     # = z = net = x w + x w ...
150     #
151     # The partial derivative of the total net input with respective to a given
            weight (with everything else held constant) then is:
152     # = z / w = some constant + 1 * x w ^(1-0) + some constant ...
            = x
153     def calculate_pd_total_net_input_wrt_weight(self, index):
```

```
    #Your Code Here

# An example:

nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3],
    hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55],
    output_layer_bias=0.6)
for i in range(10000):
    nn.train([0.05, 0.1], [0.01, 0.99])
    print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01, 0.99]]]), 9))
```

3 Tasks

- Given the training set `horse-colic.data` and the testing set `horse-colic.test`, implement the BP algorithm and establish a neural network to predict if horses with colic will live or die. In addition, you should calculate the accuracy rate.
- Please submit a file named `E13_YourNumber.pdf` and send it to `ai_2018@foxmail.com`

4 Codes and Results